Performing Low-cost Electromagnetic Side-channel Attacks using RTL-SDR and Neural Networks

Pieter Robyns



Motivation and introduction



Motivation

- Information about performing EM side-channel attacks using SDR is quite scarce
 - A few academic papers, but code is often closed source
 - ChipWhisperer: open source, good info on side-channel attacks, but uses custom hardware for *power* side channels
- This talk: how to get started using RTL-SDR and open-source software
 - We'll use the EMMA framework (open source since november 2018)
- Extra: fun use case for some machine learning



Introduction: the EM side channel

- Hardware emits EM radiation during computations
 - Amplitude of emitted EM wave is proportional to power consumed
 - Some computations require more power than others
- EM side-channel attacks attempt to infer the performed computations from leaked EM radiation
- Interesting examples:
 - Operations of an encryption algorithm during a browser session
 - Key presses while typing on a keyboard
 - Memory reads / writes

Introduction: attacks in previous works

- Sniffing keystrokes from keyboard emanations
 - <u>https://www.usenix.org/event/sec09/tech/full_papers/vuagnoux.pdf</u>
- Extracting RSA / ElGamal keys from a PC
 - https://eprint.iacr.org/2015/170.pdf

- Or even CRT / LCD screens
 - <u>https://www.cl.cam.ac.uk/~mgk25/ih98-tempest.pdf</u>



Introduction: typical EM side-channel attack scenario



1. (Attacker sends plaintext to encrypt)

2. Victim inadvertently leaks EM radiation during computations

3. Attacker captures signals and infers the used encryption key through statistical analysis



Icons made by Freepik from www.flaticon.com

Correlation Electromagnetic Analysis (CEMA) on AES



Opening new horizons

• First, find out where the secret key is used



• Output of SubBytes is loaded to register \rightarrow leaks



- What happens inside the chip?
 - CPU register is in unknown initial reference state R
 - After AddRoundKey + SubBytes, the register is $D=sbox(p_s\oplus k_s)$ where s is the index of the considered key byte
- Power consumed depends on number of bit flips
 - Therefore, it's given by Hamming distance between R and D

• Hamming weight also works in practice if R = 0

• For n_m iterations (encryptions):

 $H_{sj}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus j)) - H_{s_1}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 0)) - H_{s_1}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 1)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_255}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_25}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_25}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_25}^{(1,2,\ldots,n_m)} = HW(sbox(p_s \oplus 0 \times 0 f)) - H_{s_25}^{(1,2,\ldots,n_m)} = HW(sbox$ key byte value jUse random plaintexts to increase variability in resulting Hamming weights

Simulate

leakage for

each possible



- Final step: correlate reality with model for each sample $x_t^{_{(1,2,\ldots n_m)}}$
- Highest correlation hypothesis is most likely key byte
- Absolute value of Pearson correlation
 - Note: = negative or positive linear correlation!
- "Correlation Power Attack"

$$|\rho_{x_t}, H_{sj}| = |\frac{cov(x_t, H_{sj})}{\sigma_{x_t}\sigma H_{sj}}|$$



Case study: AES CEMA attack on Arduino Duemilanove



Overview of the experiment

- 1. Measurement setup
- 2. Identifying leaking frequencies
- 3. Capturing leakage traces using RTL-SDR
- 4. Performing a standard CEMA on AES
- 5. Improving CEMA using neural networks



1. Measurement setup

- Our target: Arduino Duemilanove
 - Assuming software AES implementation black box: user supplies plaintext and the device encrypts it with an unknown key
- RTL-SDR to perform EM leakage measurements
- EM probe / directional antenna + amp
- Laptop + GNU Radio + numpy for signal processing



1. Measurement setup

- TekBox wideband amp. + probe (€ 287-331)
- Probe position: near VCC and GND pips (better quality)







1. Measurement setup



ver 2 rev 1 - 05.03.2013

UHASSELT EDM Finders
 Constraint on the second sec

2. Identifying leaking frequencies

- Next, let the device encrypt some random plaintexts at regular intervals
 - Allows us to see which frequencies leak information





2. Identifying leaking frequencies

• Let's zoom in...





3. Capturing leakage traces using RTL-SDR

• Host: using emcap from the EMMA framework:

./emcap.py --sample-rate 2000000 --frequency 70720300 --gain 20 --limit 51200 --output-dir datasets/fosdem-arduino-test rtlsdr serial

 Instruct target to perform random plaintext encryptions, but with the same key:

b1 d3 44 d0 19 ea b4 71 39 d8 3c f2 c2 02 f1 c1



3. Capturing leakage traces using RTL-SDR



3. Capturing leakage traces using RTL-SDR



• Result after 51,200 traces:

./emma.py abs 'align[15460,15680,True]' 'window[200,500]' attack fosdem-arduino-test --refset
fosdem-arduino-test --butter-cutoff 0.2 --key-low 0 --key-high 16 --max-subtasks 16

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.14 (b1)	0.06 (52)	0.06 (44)	0.03 (5f)	0.12 (19)	0.06 (eb)	0.10 (b4)	0.08 (71)	0.06 (38)	0.04 (f7)	0.07 (85)	0.06 (f3)	0.10 (c2)	0.07 (02)	0.10 (f1)	0.09 (c0)
0.12 (b0)	0.06 (99)	0.04 (f4)	0.03 (4c)	0.12 (18)	0.05 (ea)	0.10 (b5)	0.05 (aa)	0.06 (39)	0.04 (97)	0.07 (84)	0.05 (46)	0.10 (c3)	0.03 (3d)	0.10 (f0)	0.07 (c1)
0.09 (fd)	0.06 (44)	0.04 (d1)	0.03 (e2)	0.07 (54)	0.04 (3e)	0.04 (a8)	0.05 (83)	0.05 (e4)	0.04 (f6)	0.06 (a2)	0.05 (62)	0.05 (8f)	0.02 (c5)	0.05 (61)	0.06 (42)
0.08 (fc)	0.05 (42)	0.04 (f5)	0.03 (85)	0.07 (55)	0.04 (4d)	0.04 (16)	0.04 (eb)	0.05 (ba)	0.04 (dd)	0.05 (75)	0.05 (be)	0.04 (54)	0.02 (23)	0.04 (bc)	0.05 (d1)
0.08 (64)	0.05 (45)	0.04 (d0)	0.03 (2b)	0.06 (87)	0.03 (69)	0.04 (a9)	0.04 (b7)	0.05 (30)	0.03 (f0)	0.05 (3c)	0.05 (63)	0.04 (f9)	0.02 (ba)	0.04 (60)	0.04 (79)
0.08 (65)	0.05 (ef)	0.04 (ec)	0.03 (aa)	0.06 (89)	0.03 (7a)	0.04 (86)	0.04 (ea)	0.04 (71)	0.03 (d8)	0.05 (74)	0.05 (42)	0.04 (ed)	0.02 (0b)	0.04 (fa)	0.04 (f3)
0.07 (09)	0.05 (43)	0.04 (30)	0.03 (d7)	0.05 (cd)	0.03 (85)	0.04 (1b)	0.04 (78)	0.04 (26)	0.03 (94)	0.05 (89)	0.05 (c7)	0.04 (89)	0.02 (f7)	0.04 (4f)	0.04 (d0)
0.07 (b8)	0.05 (98)	0.04 (03)	0.03 (22)	0.05 (cc)	0.03 (e1)	0.04 (93)	0.04 (ba)	0.04 (9e)	0.03 (c7)	0.05 (57)	0.05 (47)	0.04 (24)	0.02 (cb)	0.04 (25)	0.04 (43)
0.07 (f0)	0.05 (ab)	0.03 (55)	0.03 (b1)	0.05 (3a)	0.03 (0b)	0.04 (37)	0.04 (7c)	0.04 (e5)	0.03 (95)	0.05 (a3)	0.05 (99)	0.04 (16)	0.02 (14)	0.04 (63)	0.04 (78)
0.07 (f1)	0.05 (20)	0.03 (52)	0.03 (12)	0.05 (56)	0.03 (7b)	0.03 (a6)	0.04 (c1)	0.04 (dc)	0.03 (f1)	0.05 (32)	0.05 (bf)	0.04 (f0)	0.02 (03)	0.04 (72)	0.04 (52)
0.07 (08)	0.05 (53)	0.03 (fb)	0.03 (39)	0.05 (eb)	0.03 (e0)	0.03 (60)	0.04 (87)	0.04 (f1)	0.03 (2f)	0.04 (ed)	0.05 (7e)	0.04 (10)	0.02 (3c)	0.04 (ba)	0.04 (61)
0.07 (ba)	0.04 (9d)	0.03 (61)	0.03 (d0)	0.05 (8f)	0.03 (4c)	0.03 (d5)	0.04 (c3)	0.03 (df)	0.03 (96)	0.04 (40)	0.05 (f2)	0.04 (52)	0.02 (90)	0.04 (c3)	0.04 (ae)
0.06 (fa)	0.04 (5a)	0.03 (f6)	0.02 (6a)	0.05 (70)	0.03 (18)	0.03 (1a)	0.04 (9a)	0.03 (ac)	0.03 (4b)	0.04 (d6)	0.05 (51)	0.04 (60)	0.02 (c7)	0.04 (9e)	0.04 (66)
0.06 (fb)	0.04 (fe)	0.03 (60)	0.02 (d9)	0.05 (0d)	0.03 (7c)	0.03 (f8)	0.04 (7a)	0.03 (5e)	0.03 (e1)	0.04 (41)	0.04 (b9)	0.04 (8e)	0.02 (97)	0.04 (bb)	0.04 (83)
0.06 (5d)	0.04 (4b)	0.03 (a6)	0.02 (8b)	0.05 (fc)	0.03 (32)	0.03 (3d)	0.04 (03)	0.03 (5f)	0.03 (03)	0.04 (56)	0.04 (57)	0.03 (41)	0.02 (62)	0.04 (62)	0.04 (2e)
0.06 (9e)	0.04 (aa)	0.03 (5c)	0.02 (66)	0.05 (36)	0.03 (75)	0.03 (87)	0.04(c9)	0.03 (b5)	0.03 (90)	0.04 (8f)	0.04 (98)	0.03 (b4)	0.02(1a)	0.04 (ec)	0.04 (15)
0.06 (5c)	0.04 (34)	0.03 (fa)	0.02 (24)	0.04 (fd)	0.03 (78)	0.03 (3c)	0.04 (a1)	0.03 (8e)	0.03 (14)	0.04 (88)	0.04 (7c)	0.03 (62)	0.02 (81)	0.04 (22)	0.04 (53)
0.06 (83)	0.04 (4a)	0.03 (h6)	0.02 (36)	0.04 (42)	0.03 (h5)	0.03 (24)	0.04 (76)	0.03 (1d)	0.03 (58)	0.04 (36)	0.04 (b8)	0.03 (50)	0.02 (c8)	0.04 (23)	0.04 (67)
0.06 (32)	0.04 (c4)	0.03 (ce)	0.02 (02)	0.04 (8d)	0.03 (54)	0.03 (52)	0.04 (9e)	0.03 (15)	0.03 (85)	0.04 (ec)	0.04 (43)	0.03 (h2)	0.02 (ea)	0.04 (21)	0.04 (dd)
0.06 (bb)	0.04(5h)	0.03 (97)	0.02 (b8)	0.04 (3b)	0.03 (f4)	0.03 (fe)	0.03 (47)	0.03 (07)	0.03(f3)	0.04 (35)	0.04 (83)	0.03 (22)	0.02 (d4)	0.03 (bd)	0.04 (74)
0.00 (00)	0.04 (JD)	0.05 (57)	0.02 (00)	0.04 (00)	0.05 (14)	1 0.05 (10)	0.05 (47)	0.05 (07)	0.05 (15)	0.04 (00)	0.04 (05)	1 0.05 (22)	1 0.02 (u+)	, 0.05 (bu)	0.04 (/4)

Predicted: b1 52 44 5f 19 eb b4 71 38 f7 85 f3 c2 02 f1 c0 Real key: b1 d3 44 d0 19 ea b4 71 39 d8 3c f2 c2 02 f1 c1



5. Improving CEMA using neural networks

- Classic CEMA side-channel attack has some issues
 - Uses only a single point \mathcal{X}_t (the one with highest correlation) from the traces
 - Requires that traces are aligned in a preprocessing step
 - Slow due to large number of points
- ML and DL to the rescue?

UHASSEL¹

- Signal can be seen as a 1D image
- Make class label for each byte value (256 classes)
- Use regular state-of-the-art image classification CNN —

 \rightarrow shown to be feasible in 2017 paper by Prouff et al. [1]

 \rightarrow similar work at Blackhat 2018 by Perin et al. [2]

Is this the best approach?

· Let's compare the input data







Is this the best approach?

- EM traces are different compared to images:
 - One training example does not give enough information to make a correct classification (assuming we target $HW(sbox(p_s \oplus k_s))$)
 - Classes are very similar to each other
 - High amounts of noise present in the data



Using neural nets to optimize sample selection

- Generate a new trace \hat{y}_t from the time-domain samples x_t
 - Combines information leaks from multiple x_t
 - Goal: approximate $HW(sbox(p_s \oplus k_s))$
 - Can be seen as dimensionality reduction

• How to determine which samples to combine and how?

 \rightarrow Optimize weights using neural networks (any architecture)



Using neural nets to optimize sample selection

Multi-Layer Perceptron (MLP)

Convolutional Neural Network (CNN)





Using neural nets to optimize sample selection

- Define loss function for one byte: $\mathcal{L}(\hat{y}_s, y_s) = 1 \frac{cov(\hat{y}_s, y_s)}{\sigma_{\hat{y}_s}\sigma_{y_s} + \epsilon}$
 - y_s is the true $HW(sbox(p_s \oplus k_s))$
 - Negative correlation: loss \rightarrow 2
 - No correlation: loss is 1
 - Positive correlation: loss $\rightarrow 0$
 - Cost function: sum of 16 loss functions (one per byte of the key)

- Implement using Tensorflow
 - Weight updates (gradients) are calculated automatically
 - We can use standard optimizers: RMSprop, ADAM, ...



Training on random keys

- Neural net needs to learn mapping between traces and Hamming weight of $sbox(p_s \oplus k_s)$
- Inputs: dataset of completely random (but known) encryptions and corresponding Hamming weights
- Using EMMA:

./emma.py abs 'align[15460,15680,True]' 'window[200,500]' corrtrain fosdem-arduino-train --valset
fosdem-arduino-test --refset fosdem-arduino-test --butter-cutoff 0.2 --key-low 0 --key-high 16



Visualization of input batch

Opening new horizons



Training on random keys



Training set cost function

Validation set cost function





Saliency after learning

Frace index

Neuron: 0 Time (samples)

./emma.py abs 'align[15460,15680,True]' 'window[200,500]' 'plot[2d]' fosdem-arduino-train --refset fosdem-arduino-test --butter-cutoff 0.2 --plot-num-traces 256 --plot-xlabel Samples --plot-ylabel Trace



1st key byte

Saliency after learning

Frace index

Neuron: 6 50 100 150 200 250 50 0 100 150 200 250 Time (samples)

./emma.py abs 'align[15460,15680,True]' 'window[200,500]' 'plot[2d]' fosdem-arduino-train --refset fosdem-arduino-test --butter-cutoff 0.2 --plot-num-traces 256 --plot-xlabel Samples --plot-ylabel Trace

UHASSELT EDM FWO Research Foundation
 Opening new horizons

7th key byte

Saliency after learning



./emma.py abs 'align[15460,15680,True]' 'window[200,500]' 'plot[2d]' fosdem-arduino-train --refset fosdem-arduino-test --butter-cutoff 0.2 --plot-num-traces 256 --plot-xlabel Samples --plot-ylabel Trace



12th key byte

Results

• Result after 51,200 traces:

./emma.py abs 'align[15460,15680,True]' 'window[200,500]' corrtest attack fosdem-arduino-train --valset fosdem-arduino-test --refset fosdem-arduino-test --butter-cutoff 0.2 --key-low 0 --key-high 16 --max-subtasks 16

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.11 (b1)	0.08 (d3)	0.08 (44)	0.06 (d0)	0.14 (19)	0.05 (ea)	0.13 (b4)	0.07 (71)	0.06 (39)	0.06 (d8)	0.06 (3c)	0.05 (f2)	0.12 (c2)	0.08 (02)	0.17 (f1)	0.08 (c1)
0.10 (b0)	0.04 (d2)	0.04 (f4)	0.02 (aa)	0.10 (18)	0.05 (eb)	0.07 (b5)	0.03 (eb)	0.05 (e4)	0.02 (d9)	0.06 (85)	0.04 (c7)	0.10 (c3)	0.03 (3d)	0.13 (f0)	0.06 (c0)
0.07 (fd)	0.03 (ef)	0.03 (f5)	0.02 (b9)	0.06 (54)	0.03 (7a)	0.05 (a9)	0.03 (e3)	0.05 (38)	0.02 (e4)	0.05 (84)	0.03 (f3)	0.05 (f9)	0.02 (14)	0.06 (61)	0.05 (42)
0.07 (fc)	0.03 (e8)	0.03 (f6)	0.02 (d5)	0.06 (87)	0.03 (3e)	0.04 (1a)	0.03 (70)	0.05 (f1)	0.02 (72)	0.04 (a2)	0.03 (47)	0.05 (24)	0.02 (f7)	0.06 (bc)	0.03 (f3)
0.06 (64)	0.03 (5a)	0.03 (d0)	0.02 (4c)	0.06 (55)	0.03 (78)	0.04 (93)	0.03 (aa)	0.04 (ba)	0.02 (41)	0.04 (74)	0.03 (b9)	0.04 (54)	0.02 (ba)	0.06 (25)	0.03 (43)
0.06 (09)	0.03 (6a)	0.03 (0f)	0.02 (6a)	0.05 (cd)	0.03 (e0)	0.04 (a8)	0.03 (a4)	0.04 (30)	0.02 (f7)	0.04 (32)	0.03 (be)	0.04 (8e)	0.02 (e7)	0.06 (72)	0.03 (6f)
0.06 (65)	0.03 (e9)	0.03 (61)	0.02 (e7)	0.05 (3a)	0.02 (e1)	0.03 (2f)	0.03 (2c)	0.04 (e5)	0.02 (e3)	0.04 (56)	0.03 (f8)	0.04 (8f)	0.02 (82)	0.06 (4f)	0.03 (79)
0.06 (5c)	0.03 (99)	0.02 (28)	0.02 (53)	0.05 (89)	0.02 (7b)	0.03 (1b)	0.03 (56)	0.03 (ca)	0.02 (cb)	0.04 (57)	0.03 (62)	0.04 (89)	0.02 (3c)	0.06 (60)	0.03 (d1)
0.05 (08)	0.03 (ae)	0.02 (0d)	0.02 (7a)	0.05 (cc)	0.02 (85)	0.03 (92)	0.03 (b7)	0.03 (dc)	0.02 (96)	0.04 (d6)	0.03 (46)	0.04 (ed)	0.02 (f1)	0.05 (fa)	0.03 (2e)
0.05 (b8)	0.03 (42)	0.02 (b6)	0.02 (6c)	0.05 (eb)	0.02 (4d)	0.03 (37)	0.03 (bb)	0.03 (9e)	0.02 (53)	0.04 (75)	0.03 (99)	0.04 (10)	0.02 (23)	0.05 (ba)	0.03 (66)
0.05 (ba)	0.03 (c5)	0.02 (bd)	0.02 (bf)	0.05 (56)	0.02 (74)	0.03 (3d)	0.03 (11)	0.03 (5f)	0.02 (1a)	0.04 (a8)	0.03 (42)	0.04 (60)	0.02 (d9)	0.05 (ec)	0.03 (e5)
0.05 (2d)	0.03 (3e)	0.02 (03)	0.02 (d7)	0.05 (70)	0.02 (41)	0.03 (e7)	0.03 (8e)	0.03 (69)	0.02 (ef)	0.04 (8f)	0.03 (e7)	0.04 (62)	0.02 (08)	0.05 (63)	0.03 (dc)
0.05 (f1)	0.03 (c4)	0.02 (f0)	0.02 (85)	0.04 (91)	0.02 (55)	0.03 (2e)	0.02 (ae)	0.03 (71)	0.02 (4a)	0.04 (89)	0.03 (7e)	0.04 (52)	0.02 (80)	0.05 (73)	0.03 (0d)
0.05 (f0)	0.02 (ff)	0.02 (fd)	0.02 (a3)	0.04 (38)	0.02 (69)	0.03 (09)	0.02 (2d)	0.03 (26)	0.02 (31)	0.04 (0d)	0.03 (63)	0.04 (f0)	0.02 (40)	0.05 (4e)	0.03 (d7)
0.05 (b9)	0.02 (44)	0.02 (94)	0.02 (36)	0.04 (fd)	0.02 (4c)	0.03 (a6)	0.02 (a5)	0.03 (df)	0.02 (16)	0.04 (3d)	0.03 (7d)	0.04 (4f)	0.02 (a5)	0.05 (c3)	0.03 (f2)
0.05 (fb)	0.02 (55)	0.02 (fa)	0.02 (ce)	0.04 (3b)	0.02 (be)	0.03 (d0)	0.02 (36)	0.03 (07)	0.02 (60)	0.04 (a3)	0.02 (1c)	0.04 (3d)	0.02 (1f)	0.04 (1b)	0.03 (d0)
0.05 (83)	0.02 (43)	0.02 (9f)	0.02 (f3)	0.04 (8f)	0.02 (08)	0.03 (8a)	0.02 (97)	0.03 (bb)	0.02 (65)	0.03 (41)	0.02 (98)	0.04 (16)	0.02 (ac)	0.04 (49)	0.03 (83)
0.05 (32)	0.02 (34)	0.02 (30)	0.02 (0b)	0.04 (86)	0.02 (e3)	0.03 (f9)	0.02 (e2)	0.03 (67)	0.02 (9c)	0.03 (8a)	0.02 (c1)	0.04 (2e)	0.02 (03)	0.04 (bb)	0.03 (52)
0.05 (82)	0.02 (20)	0.02 (07)	0.02 (a4)	0.04 (0f)	0.02 (62)	0.03 (87)	0.02 (08)	0.03 (1d)	0.02 (62)	0.03 (0c)	0.02 (51)	0.03 (41)	0.02 (fd)	0.04 (21)	0.03 (38)
0.05(5a)	0.02 (79)	0.02 (fc)	0.02 (4d)	0.04 (90)	0.02 (ae)	0.03 (39)	0.02 (db)	0.03 (31)	0.02 (95)	0.03 (53)	0.02 (7c)	0.03 (h2)	0.02 (20)	0.04 (24)	0.03 (dd)
0.05 (54)	0.02 (757]	0102 (10)	0102 (10)	0.01 (20)	0102 (00)	1 0105 (55)	1 0102 (00)	0.05 (51)	1 0102 (33)	1 0105 (55)	1 0102 (70)	1 0105 (02)	1 0102 (20)	1 0101 (21)	1 0105 (44) 1

Predicted: b1 d3 44 d0 19 ea b4 71 39 d8 3c f2 c2 02 f1 c1 Real key: b1 d3 44 d0 19 ea b4 71 39 d8 3c f2 c2 02 f1 c1



Results from my paper

- <u>https://tches.iacr.org/index.php/TCHES/article/view/7332</u>
- Comparison to state-of-the-art CNN (2017)



19-layer CNN trained with avg. cross-entropy loss



Figure 8: Mean rank based on a 10-fold cross-validation of the frequency-domain two-layer MLP CO model in relation to the number of validation traces used for a CEMA attack. The correct key is guessed for each dataset after approximately 1,000 traces. Adding more traces to the validation set further increases the mean confidence to 0.099, 0.092, and 0.066 for respectively the ASCAD_desync50 and ASCAD_desync100 datasets.

UHASSELT

Figure 12: Mean rank based on a 10-fold cross-validation of the best_cnn model from the work of Prouff et al. [38], in relation to the number of validation traces used for a classification attack. Their model is able to determine the correct key for the ASCAD dataset after 1,910 traces, but performs poorly on the ASCAD_desync50 and ASCAD_desync100 datasets compared to our best CO models.

Conclusions

- Spurious EM emanations leak information about the state of a device
- Performing a CEMA attack using a low-cost RTL-SDR is feasible against an Arduino running software AES
 - Unknown key found after ~51,200 traces
- Neural networks can be trained to improve sample detection / remove noise from EM traces
 - Improves results of CEMA attack

Questions? pieter.robyns@uhasselt.be



Extra slides



Better measurement



UHASSELT EDM FWO Research Foundation
Penders
Opening new horizons

Differences between CO and avg. cross-entropy optimization

Correlation optimization

- Always uses 16 output neurons
- Calculates correlation between batch of inputs and batch of outputs instead of using an average metric for individual input / output pairs (i.e. batch size is more important)
- Not sensitive to scaling of the inputs (correlation is independent of scale)
- In practice (for the ASCAD benchmark dataset), we obtain better results in shorter time with a much shallower network

Average cross-entropy optimization

- Two possibilities: predict key (256 classes) or predict Hamming weight of sbox (9 classes)
- ➤ 256 classes
 - Problem: single trace does not contain enough information to predict key if only the first round of AES is considered: only the HW of sbox(p xor k) leaks here.
- 9 classes: would work for predicting the HW, but different key bytes depend on different samples of the trace. To fix:
 - Make 9 * 16 output classes (9 classes for each key byte)
 - Let network learn relation between byte index and resulting HW prediction (more complex network required)